

私のTDD

藤村大介 / @ffu_

TDDとは

"プログラムに必要な各機能について、最初にテストを書き(これをテストファーストと言う)、そのテストが動作する必要最低限な実装をとりあえず行った後、コードを洗練させる、という短い工程を繰り返すスタイル”

<http://ja.wikipedia.org/wiki/テスト駆動開発>

実演してみます

レコードバイヤーを実装します

- 最初に予算をもらう
- レコードは予算があるだけ買える

```
1 class Buyer
2   def initialize
3     ..
4 end
```

~ Twitter. See your ...

~ byflow home

~ Twitter

~ Horizontal in TopIndex

~ Fujimura Daisuke

~ Add a Text Post

~ Chrome could n...

~ Fujimura Daisuke

~ Tumblr

```
1 class Buyer
2   def initialize
3     ..
4 end
```

だめ

失敗させる(儀式)

```
tmux
~/work/my_tdd master
$ rspec spec/buyer_spec.rb

Buyer
  should fail (FAILED - 1)

Failures:

  1) Buyer should fail
     Failure/Error: false.should be_true
       expected false to be true
     # ./spec/buyer_spec.rb:7:in `block (2 levels) in <top (required)>'

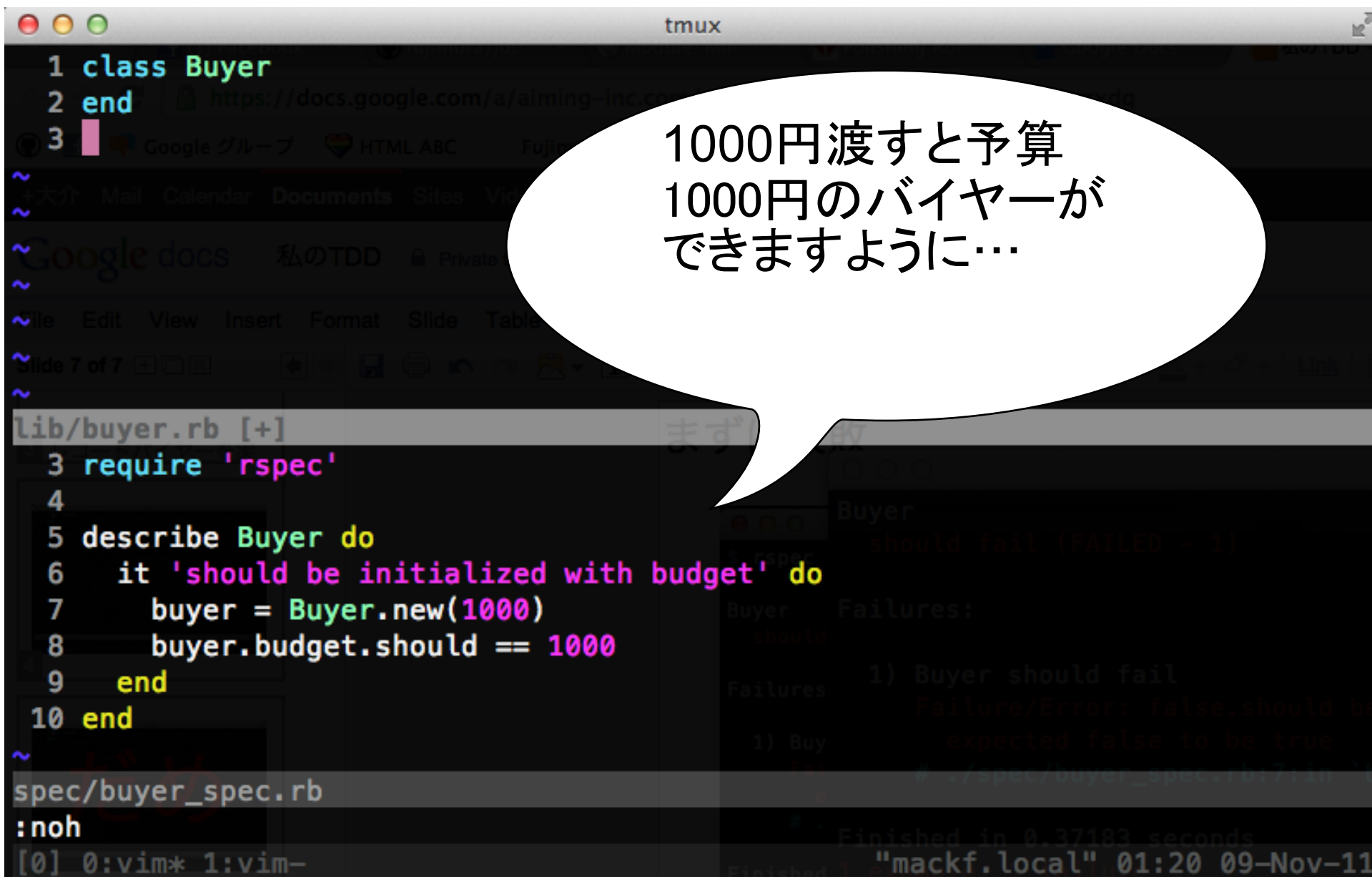
Finished in 0.37183 seconds
1 example, 1 failure

Failed examples:

rspec ./spec/buyer_spec.rb:6 # Buyer should fail
$
```

[3] 0:zsh* "mackf.local" 01:17 09-Nov-11

それから、希望する挙動を示すコードをテストとして書く



```
tmux
1 class Buyer
2 end
3
lib/buyer.rb [+]
3 require 'rspec'
4
5 describe Buyer do
6   it 'should be initialized with budget' do
7     buyer = Buyer.new(1000)
8     buyer.budget.should == 1000
9   end
10 end
spec/buyer_spec.rb
: noh
[0] 0:vim* 1:vim-
```

1000円渡すと予算
1000円のバイヤーが
できますように...

失敗する

```
tmux
$ rspec spec/buyer_spec.rb

Buyer
  should be initialized with budget (FAILED - 1)

Failures:

  1) Buyer should be initialized with budget
     Failure/Error: buyer.budget.should == 1000
     NoMethodError:
       undefined method `budget' for #<Buyer:0x007f8b4b976728>
     # ./spec/buyer_spec.rb:8:in `block (2 levels) in <top (required)>'

Finished in 0.37688 seconds
1 example, 1 failure

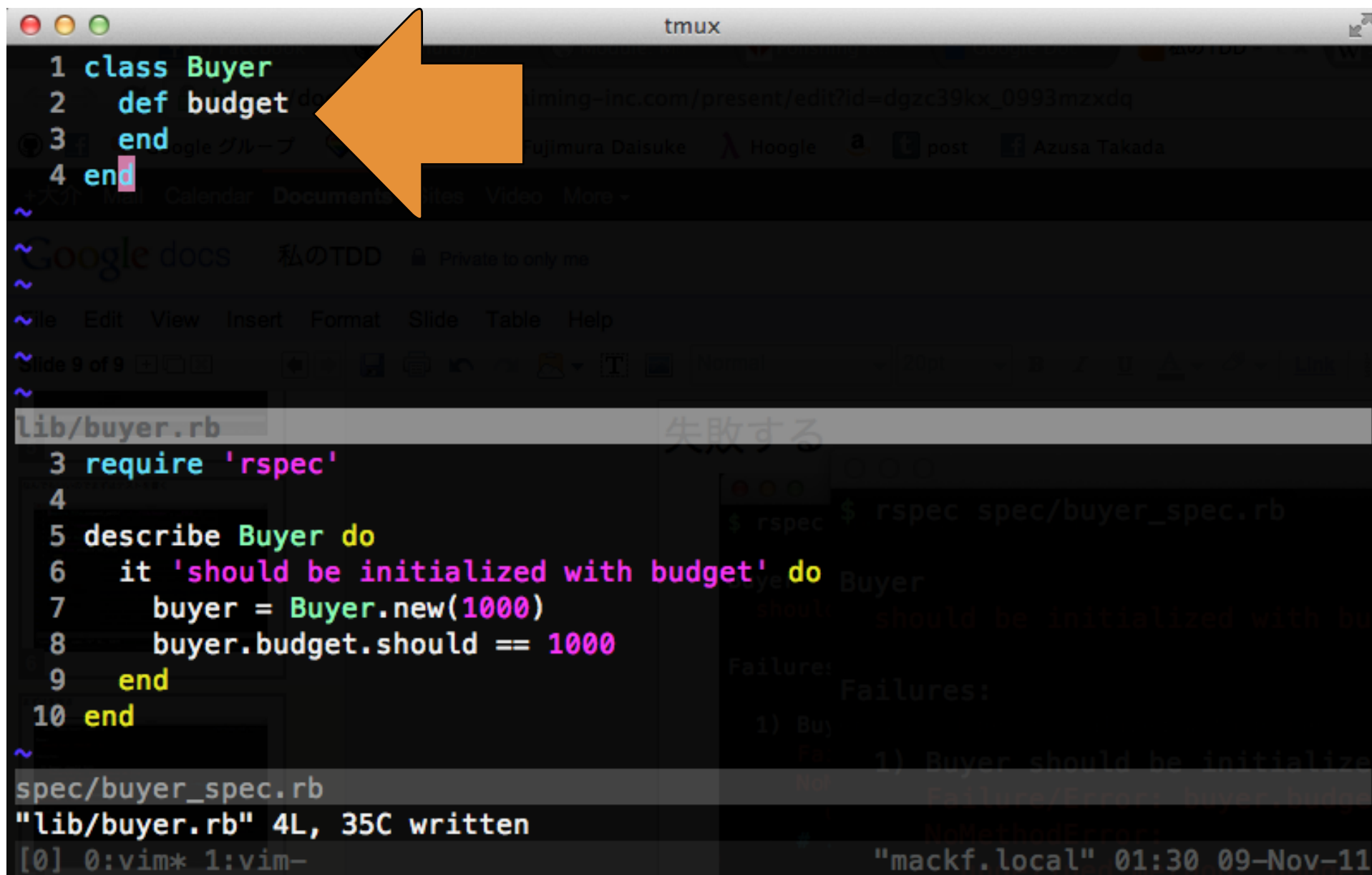
Failed examples:

rspec ./spec/buyer_spec.rb:6 # Buyer should be initialized with budget

~/work/my_tdd master
$
```

メソッドがない
そうです

Buyer#budget を実装(”がわ”だけ)



The image shows a tmux terminal window with a dark background. At the top, there are window control buttons (red, yellow, green) and the title 'tmux'. The terminal content is as follows:

```
1 class Buyer
2   def budget do
3   end
4 end
```

An orange arrow points from the right towards the 'end' on line 4.

Below this, there are several lines of '~' indicating a scrollable history. Then, the terminal shows the file 'lib/buyer.rb' with the following code:

```
3 require 'rspec'
4
5 describe Buyer do
6   it 'should be initialized with budget' do
7     buyer = Buyer.new(1000)
8     buyer.budget.should == 1000
9   end
10 end
```

Below this, there are more '~' lines, followed by the file 'spec/buyer_spec.rb' and the command:

```
"lib/buyer.rb" 4L, 35C written
[0] 0:vim* 1:vim-
```

On the right side of the terminal, there is a vertical pane showing the output of the test. It starts with '失敗する' (Failure) and shows the command '\$ rspec spec/buyer_spec.rb'. The output includes 'Buyer should be initialized with budget' and 'Failures: 1) Buyer should be initialized with budget'. At the bottom right, it shows the file path and timestamp: '"mackf.local" 01:30 09-Nov-11'.

失敗する 1000が欲しかったのにnilが返りました(当たり前)

```
tmux
~/work/my_tdd maste
$ rspec spec/buyer_spec.rb

Buyer
Buyer
  should be initialized with budget (FAILED - 1)

Failures:

  1) Buyer should be initialized with budget
     Failure/Error: buyer.budget.should == 1000
           expected: 1000
           got: nil (using ==)
     # ./spec/buyer_spec.rb:8:in `block (2 levels) in <top (required)>'

Finished in 0.37947 seconds
1 example, 1 failure

Failed examples:

rspec ./spec/buyer_spec.rb:6 # Buyer should be initialized with budget
$
~/work/my_tdd master

[3] 0:zsh* "mackf.local" 01:32 09-Nov-11
```

コンストラクタで予算を渡し、#budgetで返すようにする

(ちょっと端折りました)

```
tmux
1 class Buyer
2   def initialize(budget)
3     @budget = budget
4   end
5   def budget
6     @budget
7   end
8 end
View Insert Format Slide Table Help
Slide 12 of 12
~
lib/buyer.rb
3 require 'rspec'
4
5 describe Buyer do
6   it 'should be initialized with budget' do
7     buyer = Buyer.new(1000)
8     buyer.budget.should == 1000
9   end
10 end
~
spec/buyer_spec.rb
"lib/buyer.rb" 8L, 99C written
[0] 0:vim* 1:vim-
Date: Wed Nov 9 01:28:58 2011
commit e1fc51f60f1858232102e3d1
Author: Fujimura Daisuke <me@fu
Date: Wed Nov 9 01:14:10 2011
"mackf.local" 01:34 09-Nov-11
```

できました

```
tmux
~/work/my_tdd master
$ rspec spec/buyer_spec.rb
Buyer
Buyer#initialize(budget)
  should be initialized with budget
Finished in 0.38831 seconds
1 example, 0 failures
$

~/work/my_tdd master

[3] 0:zsh* "mackf.local" 01:36 09-Nov-110
```

というふうに

最小の粒度でテストを流して開発していきます。

普段は手動で流すのは面倒なので、ファイルの変更を監視して自動でテストを流すツール(watchrとか)を使っています。

結果を見るためにウィンドウを切り替えるのも面倒なので、常にテストの結果が画面の片隅に出ているようにしています。

ノートPCの場合はこんな感じでターミナルを重ねている

```
Terminal Shell Edit View Window Help
tmux
6
7 def buy(record)
8
9   @budget = @budget - record.price
10  @records << record
11 end
12
13 end
~/work/my_tdd master
$ rspec spec/buyer_spec.rb
Buyer
should be initialized with budget
#buy
should use budget
should not add record without enough budget (FAILED - 1)
Failures:
  1) Buyer#buy should not add record without enough budget
     Failure/Error: buyer.records.should_not include james_brown
     expected [Record<@00710a7501bed0 price=1000>] not to include #<Record<@00710a7501bed0 price=10000>
     # ./spec/buyer_spec.rb:29:in 'block (3 levels) in <top (required)>'
Finished in 0.38048 seconds
[0] 0:vim* 1:vim 2:ruby- "mackf.local" 02:40 09-Nov-11
Failed examples:
spec/buy
"lib/buy
[0] 0:vim
rspec ./spec/buyer_spec.rb:25 # Buyer#buy should not add record without enough budget
$
[3] 0:zsh*
~/work/my_tdd master
"mackf.local" 02:40 09-Nov-11
```

どうふうに1ステップずつ進みます

本番にこの状態でテストを流します。

普段は手動でテストは流すので、ファイルの変更を監視して自動でテストを流すツールをwatchとかを使っています。

結果を見るためにフィードバックを切り替えるのが遅いので、常にテストの結果が画面の外側に吐き出されるようにしています。

15 どうふうに

時は流れ

現在レコード購入機能を実装しています。これから予算制限をつけるところ。(今は無制限に買える)まずはテストを書きます

```
tmux
6
7 def buy(record)
8   @budget = @budget - record.price
9   @records << record
10 end
11
12 end
~|le Edit View Insert Format Slide Table Help
~|Slide 18 of 18
~|
lib/buyer.rb
23   buyer.budget.should == 900
24 end
25   it 'should not add record without enough budget' do
26     buyer = Buyer.new(1000)
27     james_brown = Record.new(10000)
28     buyer.buy james_brown
29     buyer.records.should_not include james_brown
30 end
31 end
spec/buyer_spec.rb
"lib/buyer.rb" 12L, 202C written
[0] 0:vim* 1:vim 2:ruby- "mackf.local" 02:11 09-Nov-11
```

予算を超えてたらレコードが増えないようにしてください

もちろん
まず失敗
させる

```
tmux
$ rspec spec/buyer_spec.rb
Buyer
  should be initialized with budget
  #buy
    should add record to his collection
    should use budget
    should not add record without enough budget (FAILED - 1)
  budget do
  Failures:

  1) Buyer#buy should not add record without enough budget
     Failure/Error: buyer.records.should_not include james_brown
       expected [#<Record:0x007f8a750fbed0 @price=10000>] not to include #<Record:0x007f8a750fbed0 @price=10000>
       Diff:
       @@ -1,2 +1,2 @@
       -#<Record:0x007f8a750fbed0 @price=10000>
       +[#<Record:0x007f8a750fbed0 @price=10000>]
       # ./spec/buyer_spec.rb:29:in `block (3 levels) in <top (required)>'

Finished in 0.38048 seconds
4 examples, 1 failure

Failed examples:

rspec ./spec/buyer_spec.rb:25 # Buyer#buy should not add record without enough budget
$
```

~/.work/my_tdd master

[3] 0:zsh* "mackf.local" 02:14 09-Nov-110

あります。これから予
するようになります。
せなら予算で分岐して
ないから

実装簡単だし、
いちいち失敗させるのはアホらしいと思いますよね？
しかし、

条件分岐を実装する際は、まずすべての分岐を通るテストを書きます

面倒かもしれませんが、どうせあとで全部手で動かすんですよ？
だったら再現できるようにしたほうがお得です。

さらに、正常系(もしくはその時動かしたい分岐)のみ通るコードを書き、他の実装に進んでしまい…アツ気がついたら異常系の想定が漏れてた、みたいなケースも起こりにくくなります。なぜならテストを書く際に全パスの処理を想定する必要があり、それを実装するようテストが示してくれるから。

(本当にやっています)

実装に戻ります

分岐を入れました。

```
tmux
6
7 def buy(record)
8   return if @budget < record.price
9
10  @budget = @budget - record.price
11  @records << record
12 end
13
14 end
~
lib/buyer.rb
23 buyer.budget.should == 900
24 end
25 it 'should not add record without enough budget' do
26   buyer = Buyer.new(1000)
27   james_brown = Record.new(10000)
28   buyer.buy james_brown
29   buyer.records.should_not include james_brown
30 end
31 end
spec/buyer_spec.rb
"lib/buyer.rb" 14L, 240C written
[0] 0:vim* 1:vim 2:ruby- "mackf.local" 02:12 09-Nov-11
```

通りました これで安心の条件分岐が実現された。

```
tmux
$ rspec spec/buyer_spec.rb ~/work/my_tdd master

Buyer
  should be initialized with budget
  #buy
    should add record to his collection
    should use budget
    should not add record without enough budget

Finished in 0.3892 seconds
4 examples, 0 failures
$ █ ~/work/my_tdd master

[3] 0:zsh* "mackf.local" 02:13 09-Nov-110
```


これを少しずつ進めていきます

ちなみにソフトウェアテストそのものの技法をしっかりと身につけておくとテストは書きやすくなり、また堅牢なコードになると思います。

この本がおすすめ

はじめて学ぶソフトウェアのテスト技法
リー・コーブランド (著), 宗 雅彦 (翻訳)



最後に、私のTDD

私、ソフトウェア開発の技法の中で何が好きか、と言うと、断然TDDです。夢中とまでは言いませんが、今更離れるのはあまりに辛いツールであります。

そんなTDDとは私にとって一体何なのだろうか？の話。

私は面倒くさがり、しかも仕事が雑です

更に落ち着きがなく、極めて散漫な性格です。仕事に時間がかかるのも嫌いです。

だからこそそのTDD。最初はウザかったけど、

- 手戻りが無い(常に再帰テストされる)
- ミスが少なくなる(事前に全分岐のテストケースを想定)
- やるべきことが示される(まず失敗させて、それから実装)
- 必要ないことをやらないで済む(テストが通れば実装完了)

と、気がついたらTDDは自分の弱点をそっとカバーしてくれるのでした。

みなさんも騙されたと思ってやってみてください。

最初は面倒かもしれませんが。慣れは必要です。

ツールや環境を整備すると、かなり負担は減ります。
開発環境でCIを動かす感覚で常にテストを流しておくとかかなり楽です。

watchrあたりを活用するとよいです。

watchr:

<https://github.com/mynymml/watchr>

終わり